Programming with Python
Part 3
Different Behavior Functions for Experiments 1-4

emerging-mind.org eJournal ISSN 2567-6466
(info@emerging-mind.org)

Gerd Doeben-Henisch
gerd@doeben-henisch.de

October 18, 2017

Contents

Abstract

According to the actual requirements we have to prepare 4 different types of behavior functions.

1 Problem to be Solved

In part 2 we have mentioned the following 4 types of behavior functions which we need:

1. The *behavior function* ϕ of the actor is 'empty' $\phi = \emptyset$. The actor functions like an 'envelope': you can see the body of the actor on the screen, but his behavior depends completely from the inputs given by a human person.

2. The *behavior function* ϕ of the actor is driven by one, fixed rule $\phi(i) = const$. The actor will do always the same, independent from the environment.

3. The *behavior function* ϕ of the actor is driven by a source of random values; therefore the output is completely random $\phi(i) = rand$.

4. The *behavior function* ϕ of the actor is driven by a source of random value but simultaneously the actor has some simple memory μ remembering the last n steps before finding food. Therefore the behavior is partially random, partially directed depending from the distance to the goal food: $\phi : I \times IS \longmapsto IS \times O$ with internal states IS as a simple memory which can collect data from the last n-many steps before reaching the goal. If the memory μ is not empty then it can happen, that the actual input maps the actual memory-content and then the memory gives the actor a 'direction' for the next steps.

In this part 3 we will program the cases 1-3 and we will implement a *food-intake function* which will increase the energy level again.

2 How to Program

After replacing all console interactions in part 2 we will now realize the four different types of behavior functions using only graphical interactions. Then we will include the 'food-intake function'.

2.1 Empty Behavior Function

The first type of required behavior is an empty *behavior function* ϕ written as $\phi = \emptyset$. In this case the actor is functioning like an 'envelope': you can see the body of the actor on the screen, but his behavior depends completely from the inputs given by a human person.

There are two options: (i) having keystrokes determine the direction or (ii) having mouse clicks directly in the grid.

Using keystrokes is a bit cumbersome, but has the great advantage in a public performance that the audience has a full control what the performer on stage are doing. Therefore we select this option for programming.

As we could see in the case of the introduction of the actor one can click into the grid and then one can easily identify the position. Thus we can copy this procedure and modify it as necessary.

For the details of all three source code files see the ZIP-Folder of the files attached to the post.

Here we discuss only the new empty behavior function with the name 'behaviorAcEmpty(e,a,win)'.

The new code starts in the 'MENTAL' part.

A text message is printed in the window to point to some neighboring cell to give a hint for a new direction. Then the window coordinates of this point with the mouse are evaluated and translated in the coordinates of the data array. Then given the old coordinates and the new coordinates the resulting new direction is computed and written in the actor object (which serves as the interface between the world managed in main() and the actor. Then the main() function computes the resulting move and computes possible obstacles.

```
def behaviorAcEmpty(e,a,win):

import numpy as np
import environment as env
import acctor as acc
import graphics as grph

#BODY PART
# Compute time since last call and change energy level

name = a.getName()
t1 = a.getTime()
t2 = e.getClock()
erate = a.getEnergyRate()
ener = a.getEnergy()

dt = t2-t1
energynew = ener - (erate * dt)
a.setEnergy(energynew)

print(PID+'Actor ',name,' has new energy level = ',energynew)

#MENTAL PART
# Determine the new direction

#Let the user click on a cell beside the actual position and determine
```

```
# the direction

message = grph.Text(grph.Point(140, 15), "Click in neighbouring cell for  NEW DIREC
message.setTextColor('red')
message.draw(win)

# Get and draw the coordinate of a point
pdir = win.getMouse()
xc=pdir.getX()
yc=pdir.getY()
print(PID+'P for new direction ',pdir,(xc,yc))

#Hide the Text
message2 = grph.Text(grph.Point(140, 15), "Click in neighbouring cell for  NEW DIREC
message2.setTextColor('HoneyDew')
message2.draw(win)

#Transform the window coordinates into the array coordinates
xs = int(xc/100)
ys = int(yc/100)
x = xs+1
y=ys+1
print(PID+'P for new direction in Array  ',(x,y))

#compute the resulting direction rDir
#Getting the old position

p=a.getPos()
print(PID+'Old Position of Actor = ',p)
xold=p[0]
yold=p[1]

if xold ==  x:
print(PID+'xold = x points to dir 1,3')
if yold < y:
dirnew = 3
print(PID+'dir = ',dirnew,(xold,yold),(x,y))
else:
dirnew = 1
print(PID+'dir = ',dirnew,(xold,yold),(x,y))

elif yold == y:
print(PID+' yold = y points to dir 2,4')
if xold < x:
```

```
dirnew = 2
print(PID+'dir = ',dirnew,(xold,yold),(x,y))
else:
dirnew = 4
print(PID+'dir = ',dirnew,(xold,yold),(x,y))
else:
print(PID+'ERROR with DIRECTION !!!')
dirnew = a.getDir()
print(PID+'Error Dirnew ',dirnew)

a.setDir(dirnew)
print(PID+'New direction of actor ',name, ' is = ',dirnew)

a.setOutputMessage('yesMove')
print(PID+'actor ',name, 'wants to move')
```

The following text shows the beginning and the end of the console printouts. It ends with the dead of the actor because there is not yet a food-intake function implemented.

All printouts have in the beginning acronyms 'ACC', 'MAIN' or 'ENV' pointing to the file where they have been printed. This is helpful for 'logical debugging'.

The acronyms are automatically generated by using a string variable at the beginning of a file (like *PID="MAIN: "*) which is included in every print-statement like *print(PID+ ...)*.

```
MAIN: P  Point(336.0, 67.0) (336.0, 67.0)
MAIN: Center Array   (4, 1)
MAIN: MAP OF ARRAY

MAIN: [[0 0 0 1 1 1 0]
[1 0 0 1 0 2 0]
[1 0 0 0 0 0 0]
[3 0 0 0 0 1 0]
[0 0 0 0 0 0 0]
[0 0 0 0 0 0 0]
[0 0 0 0 0 1 0]]
MAIN: ATTENTION: Columns represent Rows on screen!

ACC: Actor  A1  has new energy level =  2000
ACC: P for new direction  Point(336.0, 161.0) (336.0, 161.0)
```

```
ACC: P for new direction in Array   (4, 2)
ACC: Old Position of Actor =  (4, 1)
ACC: xold = x points to dir 1,3
ACC: dir =  3 (4, 1) (4, 2)
ACC: New direction of actor  A1  is =  3
ACC: actor  A1 wants to move
MAIN: New Position planned =  3
MAIN: Actual Position =  (4, 1)
ENV: Inside Newposition - dir,xold,yold  3 4 1
ENV: dir 3
MAIN: New Position planned =  (4, 2)
MAIN: actor = A1  will move
ENV: Type = 3  has moved to (4, 2)
MAIN: actor = A1  has moved
MAIN: Environment Time = 0
MAIN: Actual Environment Time =  0
MAIN: New  Environment Time =  1
ACC: Actor  A1  has new energy level =  1998
...

ACC: Actor  A1  has new energy level =  -70
ACC: P for new direction  Point(267.0, 631.0) (267.0, 631.0)
ACC: P for new direction in Array   (3, 7)
ACC: Old Position of Actor =  (2, 7)
ACC:  yold = y points to dir 2,4
ACC: dir =  2 (2, 7) (3, 7)
ACC: New direction of actor  A1  is =  2
ACC: actor  A1 wants to move
MAIN: New Position planned =  2
MAIN: Actual Position =  (2, 7)
ENV: Inside Newposition - dir,xold,yold  2 2 7
ENV: dir 2
MAIN: New Position planned =  (3, 7)
MAIN: actor = A1  will move
ENV: Type = 3  has moved to (3, 7)
MAIN: actor = A1  has moved
MAIN: Environment Time = 45
MAIN: Actual Environment Time =  45
MAIN: New  Environment Time =  46
MAIN: ATTENTION: Actor  A1  has no more Energy!!!
MAIN: THE GAME SAYS GOODBYE!
```

2.2 Fixed Behavior Function

If an actor shows a fixed behavior $\phi(i) = const$ then the actor will act always in the same way, independent from the environment.

For this case many variants are conceivable. We select here the case, that the actor continues to go in one direction until he hits an obstacle. In this case he will change the direction either in a right-hand manner +1 or in a left-hand manner -1, but not alternating. Thus we have a *left-hand type* of an actor or a *right-hand type*. We select here the right-hand type (clockwise).

To enable such a fixed behavior the actor needs a *minimal feedback* from the environment. Such a minimal feedback is already provided by the environment manager through the mechanism of the *Input-Message in the actor object*. Every time the environment manager detects an obstacle then he writes this as a feedback into the InputMessage buffer.

Thus the actor gives as his new direction always the old direction as long as the getInputMessage() does not show the value 'NoMove'. If this feedback shows up then the actor has to change his direction in a fixed manner.

```
def behaviorActFix(e,a):

import numpy as np
import environment as env
import acctor as acc
import math

#BODY PART
# Compute time since last call and change energy level

name = a.getName()
t1 = a.getTime()
t2 = e.getClock()
erate = a.getEnergyRate()
ener = a.getEnergy()

dt = t2-t1
energynew = ener - (erate * dt)
a.setEnergy(energynew)
```

```
print(PID+'Actor ',name,' has new energy level = ',energynew)

#MENTAL PART
#Determining the new direction

dirnew = a.getDir()

#Checking for feedback. If 'NoMove' then change direction

f = a.getInputMessage()
print(PID+'Input Message from ENV  ',f)

if f == "NoMove":
print(PID+'Identification of -NoMove- happened')
dirold = a.getDir()
dirnew = dirold + 1
if dirnew > 4:
dirnew = dirnew % 4
print(PID+'dirold, dirnew if >4 ',dirold, dirnew)

# Determine the new direction

a.setDir(dirnew)
print(PID+'New direction of actor ',name, ' is = ',dirnew)

a.setOutputMessage('yesMove')
print(PID+'actor ',name, 'wants to move')
```

The beginning of the console log-data shows how the program works.

```
MAIN:
P  Point(150.0, 349.0) (150.0, 349.0)
MAIN:
Center Array   (2, 4)
MAIN:
MAP OF ARRAY

MAIN:
[[0 0 0 0 1 0 1]
[1 0 0 3 1 0 0]
[0 2 0 1 0 0 0]
[0 0 0 0 0 0 0]
[0 0 0 0 0 0 0]
[0 0 0 1 0 0 0]
```

```
[0 0 0 1 1 0 0]]
MAIN:
ATTENTION: Columns represent Rows on screen!

ACC:
Actor  A1  has new energy level =  2000
ACC:
Input Message from ENV   NoMove
ACC:
Identification of -NoMove- happened
ACC:
New direction of actor  A1  is =  2
ACC:
actor  A1 wants to move
MAIN:
New Position planned =  2
MAIN:
Actual Position =  (2, 4)
ENV:
Inside Newposition - dir,xold,yold  2 2 4
ENV:
dir 2
MAIN:
New Position planned =  (3, 4)
MAIN:
actor = A1  has hit an obstacle
MAIN:
Environment Time = 0
MAIN:
Actual Environment Time =  0
MAIN:
New  Environment Time =  1
ACC:
Actor  A1  has new energy level =  1998
ACC:
Input Message from ENV   NoMove
ACC:
Identification of -NoMove- happened
ACC:
New direction of actor  A1  is =  3
ACC:
actor  A1 wants to move
MAIN:
New Position planned =  3
```

```
MAIN:
Actual Position =  (2, 4)
ENV:
Inside Newposition - dir,xold,yold  3 2 4
ENV:
dir 3
MAIN:
New Position planned =  (2, 5)
MAIN:
actor = A1  has hit an obstacle
MAIN:
Environment Time = 1
MAIN:
Actual Environment Time =  1
MAIN:
New  Environment Time =  2
ACC:
Actor  A1  has new energy level =  1994
ACC:
Input Message from ENV   NoMove
ACC:
Identification of -NoMove- happened
ACC:
New direction of actor  A1  is =  4
ACC:
actor  A1 wants to move
MAIN:
New Position planned =  4
MAIN:
Actual Position =  (2, 4)
```

2.3 Random Behavior Function

The random behavior function is driven by a source of random values; therefore the output is completely random $\phi(i) = rand$.

To adapt the program for this version it is only necessary to rewrite the actor function of the file 'acctor.py' as 'behaviorAcRand(e,a)'.

As You can see in the source code below one needs only a *random generator* for one of the four possible directions {1, ..., 4}

```
def behaviorAcRand(e,a):
```

```
import numpy as np
import environment as env
import acctor as acc

#BODY PART
# Compute time since last call and change energy level

name = a.getName()
t1 = a.getTime()
t2 = e.getClock()
erate = a.getEnergyRate()
ener = a.getEnergy()

dt = t2-t1
energynew = ener - (erate * dt)
a.setEnergy(energynew)

print(PID+'Actor ',name,' has new energy level = ',energynew)

#MENTAL PART
# Determine the new direction

dirspace = 4
dirnew = np.random.randint(1,dirspace+1) <----- Random generator
a.setDir(dirnew)
print(PID+'New direction of actor ',name, ' is = ',dirnew)

a.setOutputMessage('yesMove')
print(PID+'actor ',name, 'wants to move')
```

2.4 Food-Intake Function

The idea of the food-intake function is as follows: if an actor has reached in the grid a cell immediately besides a food cell which has a border in common than we assume (in the simple case) that the actual level of energy will be increased by a certain pre-defined amount of energy associated with the food. In later versions this can become much more sophisticated.

The mechanism is a an interaction between the environment manager and the actor. If the environment manager recognizes a food-object immediately besides an actor than the environment manager writes in the 'intake' Buffer from the actor, the amount of energy, which is associated with the food-intakte. It depends then from the actor, whether he will take

the notice and uses it. The normal way will be – as assumed here – that the actor will take the food and increases thereby his energy level.

To enable this new interaction between actor and environment we have changed the class acctor.py a bit (see for the full code the attached zip-file).(See for instructions to python classes the python tutorial [pyt17])

We have generated a small list of internal variables, which can be reached by class methods:

```
class ACTOBJ:
"""This is a first sketch for an actor class. It has to be developed further """

type = 'actor'
inpMsg =''      #Message from the environment
OutMsg = ''     #Message to the environment
intake = 0      #Energy Offer with food
...
    def setInputMessage(self,inputMessage):
#Some message to the environment
self.inpMsg=inputMessage

def getInputMessage(self):
return  self.inpMsg

def setIntake(self,intake):
#Some message from the environment to the actor
self.intake=intake

def getIntake(self):
return  self.intake
```

Then we have added to the fixed behavior function 'behaviorActFix(e,a)' a passage where the actor checks whether there is some possible food-intake nearby. If yes, then he adds this offered energy to his energy level and sets the intake parameter back to zero.

```
def behaviorActFix(e,a):

import numpy as np
import environment as env
import acctor as acc
import math
```

```
name = a.getName()

#BODY PART
#INTAKTE OF FOOD IF AVAILABLE

f = a.getIntake()
if f > 0:
eold = a.getEnergy()
enew = eold + f
a.setEnergy(enew)
a.setIntake(0)      #Set value back to zero
print(PID+'Actor ',name,' has  increased ENERGYLEVEL = ',enew)
```

On the part of the environment manager there has to be changed only the section where the actor hits an obstacle. If this obstacle will be recognized as 'FOOD' then the environment manager will give the actor an intake-message which offers the amount of energy which can be part of the actor, if he wants.

```
Err, xnew, ynew = env.newPosition(dirnew,xold,yold)

if Err == -1:
print(PID+'Err = ', Err)
else:
print(PID+'New Position planned = ', (xnew,ynew))

if gr2[xnew-1,ynew-1] != 0:
#Telling the actor that the planned new position is occupied
a1.setInputMessage("NoMove") <----- Obstacle
print(PID+'actor =',name,' has hit an obstacle at ',xnew,ynew)
if gr2[xnew-1,ynew-1] == 2:
print(PID+'actor =',name,' has hit FOOD')
a1.setIntake(1000) <----- Recognition of Food

else:
type=3 #actor
#First change position in data array
print(PID+'actor =',name,' will move in Array',xnew,ynew)
env.moveArray(type,xold,yold,xnew,ynew,gr2)
#Tell the actor his new position
a1.setPos(xnew,ynew)
#Next change position in window
```

```
print(PID+'Inserting actor at', xnew,ynew)
env.introduceActor(xnew,ynew,dir,name,color,energy,distance,win)
env.deleteActor(p,win)
a1.setInputMessage('YesMove')
print(PID+'actor =',name,' has moved')
else:
print(PID+'No move wanted by actor.')
```

The text below shows the beginning of a console-log text. There was an array with food in the 3rd row at (0,2) in the data array and (1,3) in the window grid. The actor started at the (0,6) array position (i.e. (1,7) window grid position). The actor moved towards the food, got the message from the environment manager that there is food-presence with '1000' energy points, the actor recognized this and increased his energy level by this amount.

```
MAIN:
P  Point(59.0, 652.0) (59.0, 652.0)
MAIN:
Center Array   (1, 7)
MAIN:
MAP OF ARRAY

MAIN:
[[0 0 2 0 0 0 3]
[0 0 0 0 0 1 0]
[0 0 0 0 0 0 0]
[1 0 1 0 0 1 0]
[0 0 0 0 0 0 0]
[0 1 0 0 0 0 0]
[1 0 0 0 0 0 0]]
MAIN:
ATTENTION: Columns represent Rows on screen!

ACC:
Actor  A1  has new energy level =  2000
ACC:
Input Message from ENV
ACC:
New direction of actor  A1  is =  1
ACC:
actor  A1 wants to move
MAIN:
New Position planned =  1
```

```
MAIN:
Actual Position =  (1, 7)
ENV:
Inside Newposition - dir,xold,yold  1 1 7
ENV:
dir 1
MAIN:
New Position planned =  (1, 6)
MAIN:
actor = A1  will move in Array 1 6
ENV:
Type = 3  has moved to (1, 6)
MAIN:
Inserting actor at 1 6
MAIN:
actor = A1  has moved
MAIN:
Environment Time = 0
MAIN:
Actual Environment Time =  0
MAIN:
New  Environment Time =  1
ACC:
Actor  A1  has new energy level =  1998
ACC:
Input Message from ENV   YesMove
ACC:
New direction of actor  A1  is =  1
ACC:
actor  A1 wants to move
MAIN:
New Position planned =  1
MAIN:
Actual Position =  (1, 6)
ENV:
Inside Newposition - dir,xold,yold  1 1 6
ENV:
dir 1
MAIN:
New Position planned =  (1, 5)
MAIN:
actor = A1  will move in Array 1 5
ENV:
Type = 3  has moved to (1, 5)
```

```
MAIN:
Inserting actor at 1 5
MAIN:
actor = A1  has moved
MAIN:
Environment Time = 1
MAIN:
Actual Environment Time =  1
MAIN:
New  Environment Time =  2
ACC:
Actor  A1  has new energy level =  1994
ACC:
Input Message from ENV   YesMove
ACC:
New direction of actor  A1  is =  1
ACC:
actor  A1 wants to move
MAIN:
New Position planned =  1
MAIN:
Actual Position =  (1, 5)
ENV:
Inside Newposition - dir,xold,yold  1 1 5
ENV:
dir 1
MAIN:
New Position planned =  (1, 4)
MAIN:
actor = A1  will move in Array 1 4
ENV:
Type = 3  has moved to (1, 4)
MAIN:
Inserting actor at 1 4
MAIN:
actor = A1  has moved
MAIN:
Environment Time = 2
MAIN:
Actual Environment Time =  2
MAIN:
New  Environment Time =  3
ACC:
Actor  A1  has new energy level =  1988
```

```
ACC:
Input Message from ENV   YesMove
ACC:
New direction of actor  A1  is =  1
ACC:
actor  A1 wants to move
MAIN:
New Position planned =  1
MAIN:
Actual Position =  (1, 4)
ENV:
Inside Newposition - dir,xold,yold  1 1 4
ENV:
dir 1
MAIN:
New Position planned =  (1, 3)
MAIN:
actor = A1  has hit an obstacle at  1 3
MAIN:
actor = A1  has hit FOOD
MAIN:
Environment Time = 3
MAIN:
Actual Environment Time =  3
MAIN:
New  Environment Time =  4
ACC:
Actor  A1  has  increased ENERGYLEVEL =  2988
ACC:
Actor  A1  has new energy level =  2980
...
```

References

[pyt17] python. classes. 2017. https://docs.python.org/3/tutorial/classes.html?highlight=classes.